Remarks

The subject application is a CPA of Serial No. 09/226,939 (the "prior application") and will be assigned the same serial number. In the prior application, an Office Action mailed on December 15, 2000 relied upon U.S. Patent 6,108,659 to Vincent ("*Vincent*") as prior art under 35 U.S.C. 102(e) to support a rejection of the claims under 35 U.S.C. § 103(a). On June 13, 2001 an interview was conducted between Applicants' representative, Brent Folsom, and Examiners Anh Ly and Hosain Alam. Applicants would like to thank Examiners Ly and Alam for the courtesy and assistance extended during the interview. During the interview, the reliance on *Vincent* in the rejection under 35 U.S.C. § 103(a) in the Office Action and the applicability of 35 U.S.C. §103(c) to the prior application were discussed. At that time, an agreement was reached that in the event Applicants filed a CPA, the applicability of 35 U.S.C. §103(c) to the subject application would be considered.

Pursuant to the agreement reached, Applicants file a CPA herewith. Further, Applicants submit that *Vincent* (U.S. Patent 6,108,659) and the prior application (Serial No. 09/226,939) were, at the relevant time, commonly owned by Platinum Technology IP, Inc. or subject to an obligation of assignment to Platinum Technology IP, Inc. Therefore, pursuant to 35 U.S.C. §103(c), *Vincent* does not qualify as prior art and cannot preclude the patentability of the claims of the subject application.

In addition, claims 1-30 have been amended to correct certain informalities and several typographical errors. Applicants respectfully submit that the claim amendments are not submitted in response to any claim rejections and are not submitted for purposes of patentability.

In view of the above Remarks, allowance of claims 1-30 is respectfully requested. The Commissioner is authorized to charge any underpayment or credit any overpayment to Deposit Account No. 06-1325 for any matter in connection with this response, including any fee for extension of time, which may be required.

Respectfully submitted,

Date: 6/15/01

By: Brent A. Folsom

Brent A. Folsom
Reg. No. 43,614

FLIESLER, DUBB, MEYER & LOVEJOY LLP
Four Embarcadero Center, Suite 400
San Francisco, California 94111-4156
Telephone: (415) 362-3800

APPENDIX

As required by 37 C.F.R. § 1.121(c)(ii), marked-up copies of the claims amended in this Response are provided below with insertions underlined and deletions bracketed:

1. (Amended) A method of automatically generating a complete dependency graph of a database code object, using a computer which has a processor, memory, display, input/output devices and connectivity software to connect to a database server, the method comprising [the steps of]:

applying a recursive algorithm that queries a database for dependency information and outputs a direct dependency graph of a database code object, the "direct dependency graph" containing dependencies representing dependencies that do not involve dependencies on triggers and on implementations of object oriented [object-oriented] code objects in the database;

applying the recursive algorithm on each of the object oriented [object-oriented] code objects in the dependency graph to incorporate dependencies of implementations of code objects in the database;

using a parser on each of the code objects in the dependency graph to identify DML statements that "fire" ["fires"] triggers [thereby identifying] so as to identify dependencies on triggers;

applying the recursive algorithm on each of the triggers to incorporate the dependencies of the triggers into the dependency graph; and

repeating the procedure for incorporating dependencies of implementation of object oriented code objects as well as the dependencies on triggers and its dependencies until new dependencies are not added to the dependency graph.

2. (Amended) The method of claim 1 further comprising [comprizing the additional step of] using the generated dependency graph to compile code objects in debug mode as part of a database code object debugging tool.

3. (Amended) The method of claim 1 further comprising [comprizing the additional step of] using the generated dependency graph to identify calling paths in a database code coverage tool.

4. (Amended) The method of claim 1 further comprising [comprizing the additional step of] using the generated dependency graph to identify calling paths in a database code object profiling tool.

5. (Amended) The method of claim 1 further comprising [comprizing the additional step of] using the generated dependency graph in a database code object testing tool.

6. (Amended) The method of claim 1 further comprising [comprizing the additional step of] using the generated dependency graph to identify dependent objects that are INVALID in the database.

7. (Amended) The method of claim 1 further comprising [comprizing the additional step of] using the generated dependency graph to identify cyclic dependencies among database code objects.

8. (Amended) The method of claim 1 further comprising [comprizing the additional step of] using the generated dependency graph in a dependency graph presentation tool.

9. (Amended) A method of generating a basic dependency tree of a code object that does not take into consideration dependencies on triggers and on implementations of object oriented [object-oriented] code objects, the method comprising [the steps of]:

querying a database catalog for direct dependencies of a code object and then for each dependency found, doing the query recursively until all basic dependencies are generated into a dependency tree.

10. (Amended) The method of claim 9 further comprising [comprizing the additional step of] using the generated dependency tree to compile code objects in debug mode as part of a database code object debugging tool.

11. (Amended) The method of claim 9 further comprising [comprizing the additional step of] using the generated dependency tree to identify calling paths in a database code coverage tool.

12. (Amended) The method of claim 9 further comprising [comprizing the additional step of] using the generated dependency tree to identify calling paths in a database code object profiling tool.

13. (Amended) The method of claim 9 further comprising [comprizing the additional step of] using the generated dependency tree in a database code object testing tool.

14. (Amended) The method of claim 9 further comprising [comprizing the additional step of] using the generated dependency tree to identify dependent objects that are INVALID in the database.

15. (Amended) The method of claim 9 further comprising [comprizing the additional step of] using the generated dependency tree to identify cyclic dependencies among database code objects.

16. (Amended) The method of claim 9 further comprising [comprizing the additional step of] using the generated dependency tree in a dependency graph presentation tool.

17. (Amended) A method of generating dependency information including dependencies of code objects on database triggers, the method comprising [the steps of]:

[1)] using a recursive algorithm for querying a database catalog for direct dependencies of a code object and then for each dependency found, doing the query recursively until all basic dependencies are generated into a dependency graph;

[2)] using a parser on each of the code objects in the dependency graph to identify DML statements that "fire" ["fires"] triggers [thereby identifying] so as to identify dependencies on triggers;

[3)] applying the recursive algorithm on each of the triggers to incorporate the dependencies of the triggers into the dependency graph; and

[4)] repeating steps 1-3 for incorporating dependencies on triggers and their dependencies until new dependencies are not added to the dependency graph.

18. (Amended) The method of claim 17 further comprising [comprizing the additional step of] using the generated dependency information to identify calling paths in a database code coverage tool.

19. (Amended) The method of claim 17 further comprising [comprizing the additional step of] using the generated dependency information to identify calling paths in a database code object profiling tool.

20. (Amended) The method of claim 17 further comprising [comprizing the additional step of] using the generated dependency information in a database code object testing tool.

21. (Amended) The method of claim 17 further comprising [comprizing the additional step of] using the generated dependency information to identify dependent objects that are INVALID in the database.

22. (Amended) A method of generating dependencies of code objects as well as implementations of object oriented code objects in a database, the method comprising [the steps of]:

[1)] applying a recursive algorithm that queries a database for dependency information and outputs a direct dependency graph of a database code object, the "direct dependency graph" containing dependencies that do not involve dependencies on triggers and on implementations of object oriented [object-oriented] code objects in the database;

[2)] applying the recursive algorithm on each of the object oriented [object-oriented] code objects in the dependency graph to incorporate dependencies of implementations of code objects in the database; and

[3)] repeating steps 1 and 2 for incorporating dependencies of implementation of object oriented code objects until new dependencies are not added to the dependency graph.

23. (Amended) The method of claim 22 further comprising [comprizing the additional step of] using the generated dependency information in the direct dependency graph to identify calling paths in a database code coverage tool.

24. (Amended) The method of claim 22 further comprising [comprizing the additional step of] using the generated dependency information in the direct dependency graph to identify calling paths in a database code object profiling tool.

25. (Amended) The method of claim 22 further comprising [comprizing the additional step of] using the generated dependency information in the direct dependency graph in a database code object testing tool.

26. (Amended) The method of claim 22 further comprising [comprizing the additional step of] using the generated dependency information in the direct dependency graph to identify dependent objects that are INVALID in the database.

27. (Amended) A system for identifying dependencies, if any, of a target data base code object, the system comprising [comprizing]:

a digital computer;

a database server coupled to the computer;

a data base coupled to the database server having data stored therein, the data including [comprizing] object oriented [object-oriented] code objects, specifications of packages, implementations of packages, specifications of types, implementations of types and triggers; and

a code mechanism for generating a dependency graph, the dependency graph being a data structure and having entries to contain representations of depending code objects, specifications of packages, implementations of packages, specifications of types, implementations of types, triggers and dependencies of triggers which are relevant to the target data base code object.

28. (Amended) A method for generating dependencies of a target data base code object, using a computer having a processor, memory, display, input/output devices, the method comprising [the steps of]:

providing a data base coupled to the computer having data stored therein, the data including [comprizing] representations of object oriented [object-oriented] code objects,

specifications of packages, implementations of packages, specifications of types, implementations of types and triggers; and

using a recursive code mechanism for generating a dependency graph, the dependency graph being a data structure and having entries to contain representations of dependent code objects, specifications of packages, implementations of packages, specifications of types, implementations of types, triggers and dependencies of triggers which are relevant to the target data base code object.

29. (Amended) A computer program product embedded on a computer readable medium for use in debugging a target data base code object comprising [comprizing]:

a recursive code mechanism for generating a dependency graph of the target data base code object the dependency graph being a data structure and having entries to contain representations of depending code objects, specifications of packages, implementations of packages, triggers and dependencies of triggers which are relevant to the target data base code object; and

a program code mechanism for using the dependency graph to debug the target data base code object.

30. (Amended) A computer program produce embedded on a computer readable medium for use in debugging a target data base code object comprising [comprizing]:

a first program code mechanism for applying a recursive algorithm that queries a database for dependency information and outputs a direct dependency graph of a database code object, the "direct dependency graph" containing dependencies representing

dependencies that do not involve dependencies on triggers and on implementations of object oriented [object-oriented] code objects in the database;

a second program code mechanism for applying the recursive algorithm on each of the object oriented [object-oriented] code objects in the direct dependency graph to incorporate dependencies of implementations of code objects in the database;

a third program code mechanism for parsing the source code of the code object to identify DML statements that "fire" ["fires"] triggers, [thereby identifying] so as to identify the dependencies on triggers;

a fourth program code mechanism for applying the recursive algorithm on each of the triggers to incorporate the dependencies of the triggers into the direct dependency graph;

a fifth program code mechanism for repeating the procedure for incorporating dependencies of implementation of object oriented code objects as well as the dependencies on triggers until new dependencies are not added to the dependency graph; and

a sixth program code mechanism for using the dependency graph to debug the target data base code object.